3

Tipo Abstracto de Dados

'' Estamos a começar a ver que o rápido avanço da técnica de programação é causado por um processo mental. Este súbito avanço está baseado mais em considerações de estilo do que em detalhes e não depende apenas da apresentação do programa mas do processo mental que o criou. Em programação, não basta ser criativo e engenhoso, é necessário também ser disciplinado e controlado para não se emaranhar na sua própria complexidade"

- Milles Harlan -

Sumário:

3.1 - Considerações Gerais

Neste capítulo tem por objectivo estudar o tipo abstracto de dados (TAD) que é uma técnica de programação para criar bibliotecas universais e desenvolver aplicações de médio e de grande porte. As linguagem de programação estrturadas (C e Pascal) implementam essa técnica com o conceito de módulo.

3.2 - Abstração em Programação

A abstração é um dos conceitos mais importantes na programação de computadores. É uma técnica de modelagem que nos permite concentrar nos aspectos essenciais de um problema, ignorando as características menos importantes.

Este conceito está presente em todas as linguagens de programação de alto nível e nos métodos empregues para programa-las. Por exemplo, o conceito de variável é uma abstracção. Uma variável é um conceito abstracto que esconde muitos aspectos técnicos que não interessam ao programador de aplicações. Quando declaramos uma variável num programa ficam escondidos as seguintes aspectos:

- A forma de representação interna;
- O número de bytes que ocupa;
- O endereço de memória onde está armazenada.

Mas para o programador, nenhuma dessas informações é importante. O que de facto interessa é poder utiliza-la para realizar um conjunto de operações que resolve um determinado problema.

3.3 - Abstração de Procedimentos

A maior parte dos elementos sintácticos de uma linguagem de programação de alto nível são abstrações. Estas abstrações facilitam o desenvolvimento de programas complexos e evitam que os programadores tenham a necessidade de manipular bits e endereços de memória.

A abstração de procedimento permite que o programador crie novas funcionalidades que escondem as sequências de instruções necessárias à sua realização.

Por exemplo, na linguagem C não existe uma instrução que seja capaz de ordenar um vector com números inteiros em ordem crescente. Para realizar essa operação, vamos criar uma função (abstração de procedimento) que vai realizar essa tarefa.

```
void  sequentialSort(int a[ ], int ultPos)
{
    int i, j;
    for ( i = 0; i <= ultPos-1; i++)
        for ( j = i+1; j <= ultPos; j++)
            if  ( a[i] > a[j] )
                troca(&a[i],&a[j]);
}
```

Após a sua implementação e posterior teste podemos utiliza-la. Nessa altura, o programador só está interresado em saber o que a função faz e não como a função realiza essa tarefa.

Numa abstração de um procedimento, o programador deve ver esse procedimento como uma caixa preta, que só mostra as seguintes informações:
Entrada: O conjunto dos dados de entrada;
Saída: O conjunto dos dados produzidos;
Objectivo: Descrição resumida sobre a sua finalidade.
Pré-Condições: Restrições necessárias para a execução desse procedimento.

3.4 - Tipo Abstracto de Dados

A abstração de dados tem os mesmos objetivos do que a abstração de procedimentos, mas está voltada para as estruturas de dados utilizadas nos programas.

A abstração de dados tem por objectivo criar novos tipos de dados e modelar o seu comportamento. Esses novos tipos são chamados de Tipos Abstratos de Dados (TAD).

Os tipos abstractos de dados são de facto, estruturas de dados que não foram previstas no núcleo das linguagens de programação e são necessárias para resolver problemas específicos. Essas estruturas estão divididas em duas partes: os dados e as operações que manipulam esses dados.

Um exemplo que ilustra esta forma de organização do software é a biblioteca stdio, que entre outras funcionalidades, define o Tipo Abstrato de Dados FILE. Se consultar o arquivo 'stdio.h', irá perceber que este contêm apenas as definições necessárias para que uma aplicação desenvolvida por terceiros consiga utilizar as variáveis do tipo FILE e as respectivas operações. Os detalhes da implementação dessas funções ficam escondidos dos programas que os programadores "comuns" desenvolveram.

O desenvolvimento de bibliotecas como a que frisamos anteriormente, baseiam-se nos seguintes princípios:

- Abstracção;
- Encapsulamento;
- Generalização.

A abstracção tem por finalidade separar os conceitos essenciais do problema a ser resolvido dos detalhes da programação.

O encapsulamento tem por finalidade manter dentro de cada módulo do sistema os detalhes que só dizem respeito e esse módulo. Cada módulo só 'exporta' as definições necessárias dos serviços que ele oferece. Essa é a base do conceito de encapsulamento das linguagens orientadas por objetos.

A generalização tem por finalidade projectar cada módulo de forma que este possa ser utilizado pelo maior número de aplicações possíveis.

No trabalho pioneiro de D. Parnas – *"On the criteria to be used in decomposing systems into modules",Communications of the ACM 15, 2, 1972, pp. 1053–1058,* são apresentados os princípios que norteiam o projetista a estruturar o software.

O desenvolvimento de aplicações que seguem esta arquitectura é normalmente mais rápido, mais seguro e mais simples. O desenvolvedor não necessita de conhecer os detalhes de implementação das bibliotecas de aplicação, ou seja, das linhas de código da implementação do TDA.

Contudo, para que o desenvolvimento de aplicações seja consistente é necessário adoptar a seguinte metodologia:

- Análise do Problema;
- Especificação do Problema.

3.5 - Análise do Problema

Vamos aplicar os conceitos vistos na secção anterior para desenvolver um programa, que irá realizar operações aritméticas de adição e multiplicação de números racionais, dados dois números inteiros. É evidentemente que o programa deverá permitir a leitura e a impressão desses números.

Em primeiro lugar, vamos extrair do problema a estrutura de dados.

```
typedf struct
 {
    int x;
    int  y;
  } TRacional;
```

Em segundo, vamos extrair as operações associadas a essa estrutura de dados.

- Ler um número inteiro;
- Criar um número racional;
- Calcular a soma de dois números racionais;
- Calcular a multiplicação de dois números racionais.

Para finalizar, vamos detalhar para cada operação, o tipo de dados que recebe, que devolve, o objectivo da operação e as condições necessárias para sua realização.

lerNumero()
Entrada: Nenhuma;
Saída : Um número inteiro;
Pré-Condições: Nenhuma.
Objectivo: Ler um número inteiro digitado pelo utilizador via teclado.

CriarRacional()
Entrada: Dois números inteiros;
Saída : Um número racional;
Objectivo: Recebe como parâmetros de entrada, dois números inteiros e devolve como retorno da função um número racional;
Pré-Condições: Nenhuma.

AdicaoRacional()
Entrada: Dois números racionais;
Saída : Um número racional;
Pré-Condições: Nenhuma;
Objectivo: Recebe como parâmetro de entrada, dois números racionais e devolve como retorno da função a soma desses números;

MultiplicacaoRacional()
Entrada: Dois números racionais;
Saída : Um número racional;
Pré-Condições: Nenhuma;
Objectivo: Recebe como parâmetro de entrada, dois números racionais e devolve como retorno da função a multiplicação desses números;

3.6 - Implementação do Problema

Cada Tipo Abstracto de Dados deve ser implementado num único módulo e cada módulo deve ser constituído por uma estrutura de dados e um conjunto de operações que o manipulam.

Para garantir que a universalidade dos módulos e a coerencia dos dados devolvidos, as operações devem implementadas como subprogramas com um sistema de controlo de erros. Com esse sistema, cada subprograma deverá devolver um código de erro à notificar o estado da operação e, com esse código, o programador saberá se operação terminou com sucesso ou não.

Para esconder as linhas de código dos subprogramas que compoêm cada módulo, cada TAD deve ser desenvolvido em dois arquivos diferentes. O arquivo interface e o arquivo com a implementação. Esses arquivos serão objecto de estudo nas próximas secções.

3.6.1- Interface de um Tipo Abstracto de Dados

No arquivo com extensão .h (header file) temos a especificação do interface, ou seja, a estrutura de dados a ser exportada e os prototipos (assinaturas) das funções que manipulam essa estrutura de dados. Esse interface é o responsável por disponibilizar os serviços do TAD aos clientes.

```
/* ------------------------------------------------------------------------------------------------
Especificação do Interface : racional.h
Objectivo: Disponibilizar operações adição e multiplicação de números racionais
------------------------------------------------------------------------------------------------*/
#ifndef RACIONAL_H_INCLUDED
#define RACIONAL_H_INCLUDED

// Tipo de Dados a ser exportado
typedef struct  racional TRacional;

// Funções exportadas
int lerNumero();
//  Retorna um numero inteiro não negativo

TRacional criar(int num, int den);
//  Recebe dois numeros inteiros e retorna um número racional

Tracional adicao(TRacional v1, TRacional  v2);
//  Recebe dois números racionais e retorna a sua adição

TRacional multiplicacao(TRacional v1, TRacional v2);
//  recebe dois números racionais e retorna a sua multiplicacão

#endif // RACIONAL_H_INCLUDED
```

Observe que a composição da estrutura de dados TRacional (struct racional) não está explicita. Dessa forma, os módulos que utilizam este TAD não poderão aceder directamente aos campos dessa estrutura (encapsulamento) e, os clientes (programadores de aplicações) só terão acesso às informações que essas funções exportam.

Em muitas aplicações, temos a necessidade de incluir um módulo dentro de um outro módulo, que por sua vez pode ser incluído num terceiro e por aí em diante. Para garantir que um cada módulo seja incluído uma e apenas uma

vez, devemos utilizar a **inclusão condicional**. Esta inclusão possui a seguinte sintaxe:

```
#ifndef  Nome_do_módulo
#define Nome_do_módulo
...
#endif
```

Observe também que comentamos apenas o comportamento de cada função. Não perdemos tempo em descrever como a função realiza essa operação. Este comentário deve ser o mais preciso possível para que os futuros clientes (programadores) possam utilizar essas bibliotecas com segurança.

Boas Practicas de Programação: *Documentar o comportamento das funções incluídas nos arquivos_interface.*

3.6.2- Implementação de um Tipo Abstracto de Dados

No arquivo com extensão .c, temos os arquivos-cabeçalho da biblioteca padrão, o arquivo do interface da biblioteca de aplicações e as linhas de código das funções que manipulam a estrutura de dados associada ao TAD. Essa inclusão deve-se ao facto de existirem definições no inteface que serão necessárias à implementação e a necessidade de garantir que as funções implementadas nesse arquivo, correspondam às funções definidas no interface.

```
/* ---------------------------------------------------------------------------------------------
Especificação das Operações: racional.c
Objectivo: Implementação das funções que manipulam um número racional
---------------------------------------------------------------------------------------------*/
include "Racional.h"

struct  racional
{
    float x;
    float y;
}

int lerNumero()
{
   int x;
   printf ("\n Entre com um numero inteiro:); scanf("%d",&x);
   return x;
}

TRacional criar( int num, int den)
{
   TRacional rac;
   rac.num = num;
```

```
    rac.den = den;
    return rac;
}

TRacional adicao(TRacional v1, TRacional v2)
{
    TRacional  res;
    res.num = (v1.num * v2.den) + (v2.num* v1.den);
    res.den = v1.den * v2.den;
    return res;
}

TRacional multiplicacao(TRacional v1, TRacional v2)
{
    TRacional res;
    res.num = v1.num * v2.num;
    res.den  = v1.den * v2.den;
    return res;
}
```

Boas Practicas de Programação: Os TAD devem ser desenvolvidos como bibliotecas de aplicação.

As funções definidas pela biblioteca-padrão são incluídas com a directiva de pré-processamento

```
#include <arquivo cabeçalho_padrão>
```

mas, as funções que nós desenvolvemos para as nossas bibliotecas, deverão ser incluídas com a directiva :

```
#include "arquivo_interface"
```

3.6.3- Aplicação Cliente

Para terminar, vamos desenvolver uma aplicação cliente muito simples. Suponhamos que necessitamos de calcular a soma e a multiplicação de dois números racionais.

Para utilizar a biblioteca de aplicações Racional.h, é necessário desenvolver um programa num arquivo separado, que iremos denominar por PRelacional.c, que possui as seguintes linhas de código.

```
/* ------------------------------------------------------------------------------------------------
Programa Cliente: PRelacional.c
Objectivo: Efectuar as operações de adição e de multiplicação de números racionais.
------------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include "Racional.h"

int main()
{
   TRacional r1, r2, r3, r4;

   r1 = criar(lernumero(), lerNumero());
   r2 = criar(lerNumero(), lerNumero());
   r3 = adicao(r1,r2);
   r4 = multiplicacao(r1, r2);

   printf("adicao : %d/%d  \n", r3.num, r3.den);
   printf("multiplicacao :%d/%d \n", r4.num, r4.den);

   return 0;
}
```

3.7 - Exercícios

3.7.1- Comente as funções definidas no TAD Trelacional.

3.7.2- Desenvolve uma aplicação que utiliza um TAD TMatriz que representa matrizes com valores reais com mxn elementos e que possui as seguintes operações:

a) Aceder: devolver o valor do elemento localizado na posição i e j da matriz;

b) Atribuir: atribuir um valor ao elemento localizado na posição i e j da matriz;

c) Adicionar: devolver uma nova matriz com o resultado da adição de duas matrizes;

d) Subtrair: devolver uma matriz que é o resultado da subtração de duas matrizes,

e) Multiplicar: devolver uma matriz que é o resultado da multiplicação de duas

matrizes;

f) Transpor: devolve uma matriz que é o resultado da transposição de uma matriz.

3.7.3- Desenvolver uma aplicação, que possui uma coleção de tipos abstractos de dados para garantir a manutenção dos dados dos voos realizados num aeroporto que essa empresa gere. Essa aplicação deve permitir realizar as consultas sobre o número de voos cancelados por cada empresa que opera nesse aeroporto.

3.7.4- Um número complexo z pode ser escrito na forma x + i y, onde x e y são números reias e i é uma unidade imaginária. Desenvolva um tipo abstracto de dados para representar um número complexo e cuja interface contenha as seguintes operações: criar, remover e aceder aos dados de um número complexo. Para além disso essa TAD deve executar as seguintes operações: adição, multiplicação, subtracção e divisão.